Why can Some Advanced Ethernet NICs Cause Packet Reordering?

Wenji Wu, Phil DeMar, Matt Crawford

***Abstract* - Intel Ethernet Flow Director is an advanced network interface card (NIC) technology. It provides the benefits of parallel receive processing in multiprocessing environments and can automatically steer incoming network data to the same core on which its application process resides. However, our analysis and experiments show that Flow Director cannot guarantee in-order packet delivery in multiprocessing environments. Packet reordering causes various negative impacts. In this paper, we use a simplified model to analyze why Flow Director can cause packet reordering. Our experiments verify our analysis.**

***Index Terms* – Packet Reordering, Flow Director, TCP, NIC.**

I. Introduction

Computing is now shifting towards multiprocessing. The fundamental goal of multiprocessing is improved performance through the introduction of additional hardware threads, CPUs, or cores (all of which will be referred to as "cores" for simplicity). The emergence of multiprocessing has brought both opportunities and challenges for TCP/IP performance optimization in such environments. Modern network stacks can exploit parallel cores to allow either message-based parallelism or connection-based parallelism as a means of enhancing performance. To date, major network stacks like Windows and Linux have been redesigned and parallelized to better utilize additional cores. While existing OSes exploit parallelism by allowing multiple threads to carry out network operations concurrently in the kernel, supporting this parallelism carries significant costs, particularly in the context of contention for shared resources, software synchronization, and poor cache efficiencies. However, investigations [1][2] indicate that CPU core affinity on network processing in multiprocessing environment can significantly reduce contention for shared resources, minimize software synchronization overheads, and enhance cache efficiency.

Core affinity on network processing has the following goals: (1) *Interrupt affinity:* Network interrupts of the same type should be directed to a single core. Redistributing network interrupts in either a random or round-robin fashion to different cores has undesirable side effects. (2) *Flow affinity*: Packets belonging to a specific TCP flow should be processed by the same core. TCP has a large and frequently accessed state that must be shared and protected when packets from the same connection are processed. Ensuring that all packets in a TCP flow are processed by a single core reduces contention for shared resources, minimizes software synchronization, and enhances cache efficiency. (3) *Network data affinity*: Incoming network data should be steered to the same core on which its application process resides. This is becoming more important with the advent of Direct Cache Access (DCA). Network data affinity maximizes cache efficiency and reduces core-to-core synchronization.

The emergence of parallel network stacks and the necessity of core affinity on network processing in multiprocessing environment require new NIC designs. An NIC should not only provide mechanisms to allow parallel receive processing to better utilize parallel network stacks, but also to facilitate core affinity on network processing in multiprocessing environments. Receive Side Scaling (RSS) [3] is a NIC technology that steps toward that direction. It supports multiple receive queues and integrates a hashing function in the NIC. The NIC computes a hash value for each incoming packet. Based on hash values, NIC assigns packets of the same data flow to a single queue and evenly distributes traffic flows across queues. With Message Signal Interrupt (MSI/MSI-X) support, each receive queue is assigned a dedicated interrupt and RSS steers interrupts on a per-queue basis. RSS provides the benefits of parallel receive processing in multiprocessing environments. Operating systems like Windows and Linux now support interrupt affinity. When an RSS receive queue (or interrupt) is tied to a specific core, packets from the same flow are steered to that core (Flow pinning). This ensures flow affinity on most OSes. However, RSS has a limitation: it cannot steer incoming network data to the same core where its application process resides. The reason is simple: the existing RSS-enabled NICs do not maintain the relationship "Traffic Flows → Network applications → Cores" in the NIC. Since network applications run on cores, we simply put is as "Traffic Flows → Cores (Applications)." This is symptomatic of a broader disconnect between existing software architecture and multicore hardware. With OSes like Windows and Linux, if an application is running on one core, while RSS has scheduled received traffic to be processed on a different core, poor cache efficiency and significant core-to-core synchronization overheads will result. The overall system efficiency may be severely degraded. To remedy the RSS limitation, the Intel Ethernet Flow Director technology [4] has been introduced. The basic idea is simple: Flow Director maintains the relationship "Traffic Flows → Cores (Applications)" in the NIC. OSes are correspondingly enhanced to support such capability. Flow Director not only provides the benefits of parallel receive processing in multiprocessing environments, it also can automatically steer packets of a specific data flow to the same core on which its application process resides, which facilitates core affinity on network processing. However, our analysis and experiments show that Flow Director cannot guarantee in-order packet delivery in multiprocessing

environments. TCP performance suffers in the event of severe packet reordering. It also makes network measurements inaccurate. In this paper, we use a simplified model to analyze why Flow Director can cause packet reordering. Our experiments verify our analysis.

II. Why Does Flow Director Cause Packet Reordering?

Intel Ethernet Flow Director supports multiple receive queues in the NIC, up to the number of cores in the system. With MSI/MSI-X and Flow-Pinning support, each receive queue has a dedicated interrupt and is tied to a specific core; each core in the system is assigned a specific receive queue. The NIC device driver allocates and maintains a ring buffer in system memory for each receive queue. For packet reception, a ring buffer must be initialized and pre-allocated with empty packet buffers. The ring buffer size is device and driver-dependent. For transport-layer traffic, Flow Director maintains a "Traffic Flow → Core" table with a single entry per flow. Each entry tracks the receive queue (core) to which a flow should be assigned. Flow Director makes use of the 5-tuple *{src_addr, dst_addr, protocol, src_port, dst_port}* in the receive direction to identify a flow in the table, and a core ID to specify the core to which the flow should be assigned. Entries within the "Traffic Flow → Core" table are updated by outgoing packets. To support Flow Director, OS must be multiple TX queue capable [5]. For an outgoing transport-layer packet, the OS records its processing core ID and passes it to the NIC to generate or update the corresponding entry within the table. The passed processing core ID advertises the core on which a network application resides. For example, a network application that resides on core ***i*** sends an outgoing packet with the header *{(src_addr: x), (dst_addr: y), (protocol: z), (src_port: p), (dst_port: q)}*. The OS records its processing core ID ***i*** and passes it to the NIC. The NIC generates or updates the corresponding flow entry in the table as *{(src_addr: y), (dst_addr: x), (protocol: z), (src_port: q), (dst_port: p), (core id: i)}*. A flow entry is deleted from the table after a configurable period of time has elapsed without traffic.

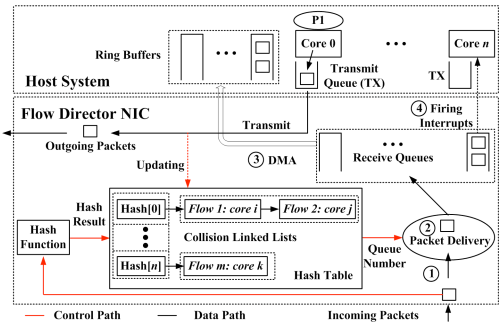

Fig. 1 Flow Director Mechanism

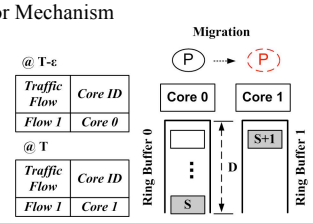

Fig. 2 Packet Reordering Analysis Model

Fig. 1 illustrates packet receive-processing for transport-layer packets with Flow Director. (1) When incoming packets arrive, the hash function is applied to the header to produce a hash result. Based on the hash result, the NIC identifies the core and hence, the associated receive queue. (2) The NIC assigns the incoming packets to the corresponding receive queues. (3) The NIC deposits via direct memory access (DMA) the received packets into the corresponding ring buffers in system memory. (4) The NIC sends interrupts to the cores associated with the non-empty queues. Subsequently, the cores respond to the network interrupts and process the received packets up through the network stack from the corresponding ring buffers one by one. As for non-Flow-Director-steering traffic, please refer to [4] for more details.

Flow Director provides the benefits of parallel receive processing in multiprocessing environments; it also facilitates core affinity on network processing. However, our analysis shows that Flow Director cannot guarantee in-order packet delivery in multiprocessing environments. TCP performs poorly with severe packet reordering [6]. In the following section, we use a simplified model to analyze why Flow Director cannot guarantee in-order packet delivery.

As shown in Fig. 2, at time $T-\varepsilon$, *Flow 1*'s flow entry maps to Core 0 in the "Traffic Flow → Core" table. At this instant, packet *S* of *Flow 1* arrives; based on the table, it is assigned to Core 0. At time T, due to process migration, *Flow 1*'s flow entry is updated and maps to Core 1. At $T+\varepsilon$, Packet *S+1* of *Flow 1* arrives and is assigned to the new core, namely Core 1. As described above, after assigning received packets to the corresponding receive queues, NIC copies them into system memory via DMA, and fires network interrupts, if necessary. When a core responds to a network interrupt, it processes received packets up through the network stack from the corresponding ring buffer one by one. In our case, Core 0 processes packet S up through the network stack from Ring Buffer 0, and Core 1 services packet S+1 from Ring Buffer 1. Let $T_{service}(S)$ and $T_{service}(S+1)$ be the times at which the network stack starts to service packets S and S+1, respectively. If $T_{service}(S) > T_{service}(S+1)$, the network stack would receive packet S+1 earlier than packet S, resulting in packet reordering. Let D be the ring buffer size and let the network stack's packet service rate be $R_{service}$ (packets/s). Assume there are *n* packets ahead of S in Ring Buffer 0 and *m* packets ahead of S+1 in Ring Buffer 1. Then, it has $T_{service}(S) = T - \varepsilon + n/R_{service}$ and $T_{service}(S+1) = T + \varepsilon + m/R_{service}$.

If ε is small and $n > m$, the condition of $T_{service}(S) > T_{service}(S+1)$ would easily hold and lead to packet reordering. Since the ring buffer size is D, the worst case is $n = D-1$ and $m = 0$, it has $T_{service}(S) = T - \varepsilon + (D-1)/R_{service}$ and $T_{service}(S+1) = T + \varepsilon$. The ring buffer size D is a design parameter for the NIC and driver. For example, the Myricom 10Gb NIC is 512, and Intel's 1Gb NIC is 256.

In a multicore system, a general-purpose OS scheduler tries to use all core resources in parallel as much as possible,

distributing and adjusting the load among the cores. Process migration across cores occurs frequently. Flow Director can easily cause packet reordering in these conditions.

III. Experiments

To validate our analysis, we ran data transmission experiments over an isolated network. A sender was directly connected to a receiver via a physical 10Gbps link.

Sender: Dell R-805; 2 Quad Core AMD Opteron 2346HE, 1.8GHz; Myricom 10G Ethernet NIC; Linux 2.6.28.

Receiver: SuperMicro Server; 2 Intel Xeon CPUs, 2.66 GHz; There are totally 4 cores in the system; Intel X520 Server Adapter with Flow Director enabled (configured with suggested default parameters [5]), 10Gbps, MTU 1500; Linux 2.6.34, Multiple TX Queue Capable.

In our experiments, iperf is used to send ***n*** parallel TCP streams from sender to receiver for 100 seconds. Iperf is a multi-threaded network application. With multiple parallel TCP data streams, a dedicated child thread is spawned and assigned to handle each stream. In our 1st experiments, we ran iperf with port *5001* in the receiver. Iperf is not pinned to a specific core. In our 2nd experiments, we pin iperfs with port 5001, 6001, 7001 and 8001 to core 0, 1, 2, and 3, respectively. Each core is sent with ***n/4*** TCP streams. In both experiments, Linux was configured to run in *multicore peak performance* mode. The receiver was instrumented to record out-of-order packets and we calculated packet reordering ratios. We also calculated the overall throughputs. The experiment results with a 95% confidence interval are shown in Table I and II.

For the 1st experiments, the degree of packet reordering is significant. At *n =200*, packet reordering ratio reaches as high as 0.897%. The experiment results validated our analysis. In the 1st experiments, since iperf is not pinned to a specific core, when the scheduler tries to distribute the load equally among the cores, it will lead to frequent process migration. Flow Director can easily cause packet reordering in these conditions. For the 2nd experiments, no packet reordering was discovered. In the 2nd experiments each flow is actually attached always to a core, there will be no process migration. As the condition of $T_{service}(S) > T_{service}(S+1)$ will not be met, Flow Director will not cause packet reordering. Therefore, reducing or avoiding process migration is a way to help Flow director to cause less/no packet reordering. Process migrations in the 1st experiments are not only degrading performance, but also causing packet reordering. In the 2nd experiments, since each flow is actually attached always to a core, there is strict core affinity on network processing (interrupt, flow, and network data affinity). Also, because there is no packet reordering, the negative effects caused by packet reordering do not exist. Therefore, the throughputs in the 2nd experiments are higher than those in the 1st experiments. In the experiments, we noticed that it is the Intel X520 10G NIC, instead of CPUs, that limits the overall bandwidth. The Intel X520 10G NIC can not reach 10Gbps when the flow director feature is turned on. Otherwise, the throughputs in the 2nd experiments would be even higher than those in the 1st experiments.

TABLE I
PACKET REORDERING RATIO

n	1st Exp.	2nd Exp.
100	0.705% ± 0.042%	0 ± 0
200	0.897% ± 0.038%	0 ± 0
500	0.635% ± 0.154%	0 ± 0
1000	0.409% ± 0.009%	0 ± 0
2000	0.129% ± 0.003%	0 ± 0

TABLE II
THROUGHPUTS (Gbps)

n	1st Exp. (Gbps)	2nd Exp. (Gbps)	Relative Incr. (%)
100	6.877 ± 0.006	6.941 ± 0.004	0.9% ± 0.097%
200	6.649 ± 0.006	6.791 ± 0.003	2.14% ± 0.12%
500	6.202 ± 0.003	6.445 ± 0.002	3.91% ± 0.012%
1000	5.943 ± 0.005	6.172 ± 0.004	3.86% ± 0.154%
2000	5.898 ± 0.008	5.909 ± 0.002	0.2% ± 0.01%

IV. Conclusion & Discussion

We use a simplified model to analyze why Flow Director can cause packet reordering in multiprocessing environments. Our experiments validate our analysis. The finding on packet reordering and our model can be applied to any other software that directs packets to the core where the application resides in the multiprocessing environments. The root cause that Flow Director can cause packet reordering is because Flow Director lacks mechanisms to ensure the satisfaction of the inequality constraint $T_{service}(S) < T_{service}(S+1)$ when it steers packets across cores. Reducing or avoiding process migration is a way to help Flow director to cause less/no packet reordering. However, this is not a true solution for the Flow Director's packet reordering problem. We believe that a possible solution is to add extra mechanisms within the NIC to ensure the satisfaction of the inequality constraint $T_{service}(S) < T_{service}(S+1)$ when Flow Director steers packets across cores.

REFERENCE

[1] J. Salehi, J. Kurose, and D. Towsley, "The effectiveness of affinity-based scheduling in multiprocessor networking," *IEEE/ACM Trans.* Netw. vol. 4, pp. 516-530, Aug. 1996.

[2] J. Hye-Churn and J. Hyun-Wook, "MiAMI: Multi-core Aware Processor Affinity for TCP/IP over Multiple Network Interfaces," in *Proc. 2009 IEEE Symposium on High Performance Interconnects*, pp. 73-82.

[3] Microsoft Corporation. (2008, November 5). *Receive-Side Scaling Enhancements in Windows Server 2008* [Online]. Available: http://www.microsoft.com

[4] Intel Corporation. (2010, November). *Intel 82599 10GbE Controller Datasheet* [Online]. Available: http://www.intel.com

[5] Intel Corporation. (2010, November 19). *IXGBE device driver README* [Online]. Available: http://www.intel.com

[6] W. Wu, P. DeMar, and M. Crawford, "Sorting reordered packets with interrupt coalescing," *computer networks,* vol. 53, no.15, pp. 2646-2662, 2009.